The Mentor-lite Prototype: A Light-Weight Workflow Management System

Jeanine Weissenfels, Michael Gillmann, Olivier Roth, German Shegalov, Wolfgang Wonner
University of the Saarland, Germany
E-mail: {weissenfels,gillmann,roth,shegalov,wonner}@cs.uni-sb.de
WWW: http://www-dbs.cs.uni-sb.de

Abstract

The Mentor-lite prototype has been developed within the research project "Architecture, Configuration, and Administration of Large Workflow Management Systems" funded by the German Science Foundation (DFG). In this paper, we present the architecture of Mentor-lite and our approach towards customizability. The demo will show the feasibility of the presented approach.

1 System Overview

The Mentor-lite prototype has evolved from the Mentor workflow management system [4, 5], but aims at a simpler architecture. The main goal of Mentor-lite has been to build a light-weight, extensible, and tailorable system with small footprint and easy-to-use administration capabilities. Our approach is to provide only kernel functionality inside the workflow engine, and consider system components like history management and worklist management as extensions on top of the kernel. The key point to retain the light-weight nature is that these extensions are implemented as workflows themselves. An invocation interface for application programs is provided by a generic IDL interface on the engine side and specific *wrappers* on the application side [3].

As shown in Figure1, the basic building block of Mentor-lite is an *interpreter* for workflow specifications. In Mentor-lite, workflows are specified in terms of state and activity charts, the specification formalism that has been adopted for the behavioral dimension of the UML industry standard and was already used in Mentor. Two additional components, the *communication manager* (*ComMgr*) and the *log manager* (*LogMgr*), are closely integrated with the workflow interpreter. All three components together form the *workflow engine*. The execution of a workflow instance can be distributed over several workflow engines at different sites. A separate *workflow log* is used at each site where a Mentor-lite workflow engine is running. Databases like the *workflow repository* (i.e., a repository of workflow specifications) or the *worklist database* can be shared by Mentor-lite workflow engines at different sites.

2 Implementation Details

The workflow specifications are interpreted at runtime, which is a crucial prerequisite for flexible

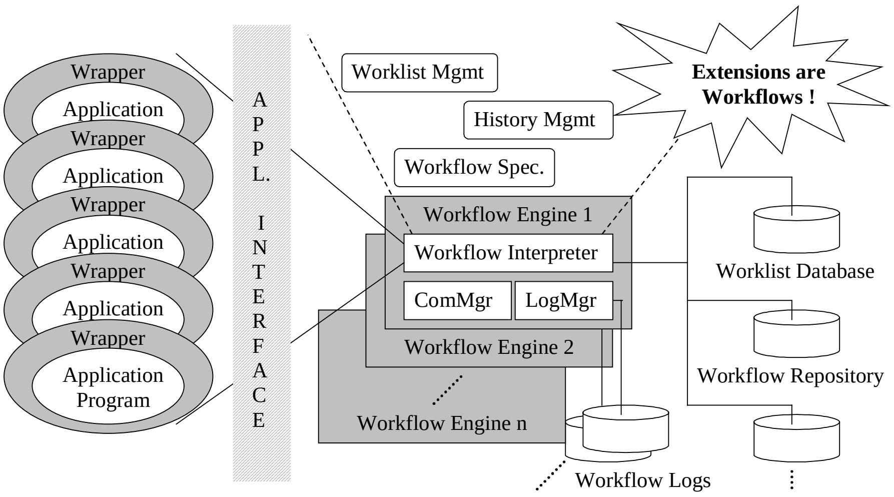

Figure 1: The Mentor-lite architecture

exception handling and dynamic modifications during runtime. The interpreter performs a stepwise execution of the workflow specification according to its formal semantics [6]. For each step, the activities to be performed by the step are determined and started.

Mentor-lite supports a protocol for distributed execution of workflows. The communication manager is responsible for sending and receiving synchronization messages between the engines. These messages contain information about locally raised events, updates of state chart variables and state information of the local engine [5]. When a synchronization message is received, the corresponding updates at the receiving site are performed. In order to guarantee a consistent global state even in the presence of site or network failures, we have built reliable message queues using the CORBA Object Transaction Services. As the basic communication infrastructure for the distributed execution we use the CORBA implementation Orbix.

The log manager provides logging facilities and recovery mechanisms. The use of a DBMS (Oracle in our implementation) for keeping the workflow log data provides advantages with regard to reliability and scalability.

The workflow engine, comprising the three components interpreter, communication manager, and log manager, is implemented as an Orbix server. Its IDL interface provides a method to start a workflow instance and a method to set variables and conditions within the workflow instance.

Application dependent facilities like *worklist management* and *history management* are implemented on top of the engine as state and activity charts. Hence, they are interpreted by the workflow interpreter just like any other workflow specification. This allows us to use the functionality of the Mentor-lite workflow engine through a single interface, namely the state chart and activity chart interpreter. The activities used in these subworkflows store worklist data and history data in an Oracle database. The user interfaces, including worklists and workhistory evaluation tools, are implemented as Java applets. The applets use the JDBC interface to access the databases.

In most other workflow management systems, interfaces to application programs have to be completely implemented inside of application wrappers. In Mentor-lite, only basic communication interfaces are implemented by wrappers, using the distributed computing environment CORBA. On top of these interfaces, protocols for complex interactions with application programs are specified in terms of state and activity charts. Hence, we use the same language for specifying workflows and interface protocols (see [3] for details). The workflow engine starts the wrappers asynchronously and uses the methods of the wrapper objects to read or set variables. The application wrappers can in turn use the workflow engine's method to set control flow variables.

3 About the Demo

For the demo, we use the specification of a simple e-commerce workflow that we developed for benchmarking workflow management systems [2]. To illustrate the complete functionality of control flow handling, the workflow includes the full spectrum of control flow structures, i.e, splits, parallelism, joins, and loops. The workflow builds on the TPC-C benchmark for transaction systems, but enhances it by control and data flow between the activities "NewOrder", "Shipment", and "Payment" and includes additional activities. To demonstrate the feasibility of our light-weight approach, the workflow includes not only automatic but also interactive activities. We will show several strategies specified as state and activity charts to manage the user's worklists.

For administration, Mentor-lite provides a Java based Workbench for workflow design, partitioning across multiple engines, and system configuration, and a Java based runtime monitoring tool. The monitoring tool is able to display the current execution state of a running workflow, highlighting the control flow path traversed so far. Further runtime data such as the current values of control flow variables are also available.

References

[1] A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.): *Workflow Management Systems and Interoperability*, NATO Advanced Study Institute, Springer, 1998

[2] M. Gillmann, P. Muth, G. Weikum, J. Weissenfels: *Benchmarking of Workflow Management Systems* (in German), German Conf. on Databases in Office, Engineering, and Scientific Applications (BTW), Freiburg, Germany, 1999

[3] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: *Integrating Light-Weight Workflow Management Systems within Existing Business Environments*, Int'l Conf. on Data Engineering (ICDE), Sydney, Australia, 1999

[4] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich: *Enterprise-wide Workflow Management based on State and Activity Charts*, in [1]

[5] P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich, G. Weikum: *From Centralized Workflow Specification to Distributed Workflow Execution*, Journal of Intelligent Information Systems, Special Issue on Workflow Management, Vol. 10, No. 2, 1998

[6] D. Wodtke, G.Weikum: *A Formal Foundation for Distributed Workflow Execution Based on State Charts*, Int'l Conf. on Database Theory (ICDT), Delphi, Greece, 1997